Capacity Planning for Vertical Search Engines

Claudine Badue[1]   Jussara Almeida[2]   Virgílio Almeida[2]
Ricardo Baeza-Yates[3]   Berthier Ribeiro-Neto[2] [4]
Artur Ziviani[5]   Nivio Ziviani[2]

[1]Federal University of Espírito Santo, Vitória, Brazil
`claudine@lcad.inf.ufes.br`

[2]Federal University of Minas Gerais, Belo Horizonte, Brazil
`{berthier,jussara,virgilio,nivio}@dcc.ufmg.br`

[3]Yahoo! Research, Barcelona, Spain & Santiago, Chile
`ricardo@baeza.cl`

[4]Google Engineering, Belo Horizonte, Brazil

[5]National Laboratory for Scientific Computing (LNCC), Petrópolis, Brazil
`ziviani@lncc.br`

Abstract

Vertical search engines focus on specific slices of content, such as the Web of a single country or the document collection of a large corporation. Despite this, like general open web search engines, they are expensive to maintain, expensive to operate, and hard to design. Because of this, predicting the response time of a vertical search engine is usually done empirically through experimentation, requiring a costly setup. An alternative is to develop a model of the search engine for predicting performance. However, this alternative is of interest only if its predictions are accurate. In this paper we propose a methodology for analyzing the performance of vertical search engines. Applying the proposed methodology, we present a capacity planning model based on a queueing network for search engines with a scale typically suitable for the needs of large corporations. The model is simple and yet reasonably accurate and, in contrast to previous work, considers the imbalance in query service times among homogeneous index servers. We discuss how we tune up the model and how we apply it to predict the impact on the query response time when parameters such as CPU and disk capacities are changed. This allows a manager of a vertical search engine to determine a priori whether a new configuration of the system might keep the query response under specified performance constraints.

Keywords: Vertical search engines, performance analysis, capacity planning model, queueing network, per-query service time imbalance, workload characterization

1 Introduction

Vertical search engines are used to search collections composed of documents of a large corporation or a subset of the Web, such as the Web of a single country or a set of documents related to a specific domain (e.g. medical information). They require a large number of computational resources to handle the incoming query traffic, which is often characterized by high peak requirements on query load. To cope with these requirements, modern vertical search engines might rely on large clusters of server machines for query processing, a configuration similar to the one observed for general Web search engines [13, 11, 32], although at a smaller scale.

In line with this trend, we consider a search engine architecture composed of a cluster of index servers, with the documents partitioned among them so that each index server stores a part of the document collection and an index for it. This architecture is usually referred to as *document partitioning* [40, 31]. The document partitioning architecture is preferred because it simplifies maintenance, simplifies the generation of the index, which can be done locally, and degrades gracefully because the failure of an index server does not prevent any query from being answered, though the final answer set might not contain all the relevant documents in the collection. The cluster also includes a broker that communicates with the various index servers. A new user query reaches the search engine through the broker, which sends a copy of the query to each index server for local processing. Afterwards, the broker receives the top ranked documents from each index server and runs a merge to determine the final set of answers to be sent to the user.

The processing of a query is split into two consecutive major phases [11]. The first phase retrieves references to the documents that contain all query terms and generates a ranking according to a relevance metric, a task usually done by the index servers. The second phase takes the merged top ranked answers and generates snippets, title, and pointer information for each of them. This task is usually performed by a cluster of document servers, each one holding a part of the document collection. While the second phase has a roughly constant cost, independent of the size of the document collection, the first phase has a cost that increases with the size of the collection. Therefore, the performance of the first phase is crucial for maintaining the scalability of modern search engines that deal with an ever-increasing number of available documents.

In this paper we propose a capacity planning methodology for vertical search engines based on estimates of the average response time for retrieving the most relevant documents for a given user query, i.e., the first and typically most costly phase of the query processing task. Our methodology is composed of two major steps: (i) the *characterization of real vertical search engine workloads*, and (ii) the *performance modeling* of a vertical search engine architecture itself. The characterization of real workloads is a key step in any capacity planning task, as it allows one to uncover key properties of the workload [25]. Here, we have analyzed and experimented with query logs from two different real-world vertical search engines, characterizing the distributions of query and term popularities, number of terms per query, query interarrival times, and query service times.

The heart of our methodology is a performance model which, in contrast to previous work (see Section 2 for further details), captures the impact of the *imbalance* in query service times among homogeneous index servers. As will be shown, this imbalance can severely degrade the average query system response time. Thus, our model provides much more accurate performance estimates than the existing literature. Moreover, driven by the assumptions of exponential query interarrival times and service times, observed in *real* workloads, our performance model is simple (but not simplistic) and relies only on data that are easy to collect using standard tools, thus being of practical use for cost-effective capacity planning decisions in real-world setups.

We demonstrate the applicability of our performance model in a number of illustrative scenarios. Once the key parameters have been estimated from easy-to-collect, small scale experimental data, our performance model can be used to investigate the behavior, provide upper-bound estimates, and ultimately drive the capacity planning decisions of a large vertical search engine. For instance, we consider a collection of 1 billion documents partitioned among 100 index servers and analyze the impact on the query system response time of adopting larger main memories, faster CPUs and disks as well as adopting application-level caching of query results at the broker. To the best of our knowledge, this is the first work to propose a methodology for capacity planning of vertical search engines.

This paper is organized as follows. Section 2 discusses research related to ours, while the architecture of our target search engine is presented in Section 3. Our capacity planning methodology for modern vertical search engine architectures is introduced in the following two sections: the characterization of key properties of real search engine workloads is presented in Section 4, and the design and validation of our performance model is given in Section 5. Section 6 shows, step-by-step, how our model can be applied in a practical capacity planning case study. Finally, conclusions and possible directions of future work are offered in Section 7.

2 Related Work

In this section, we discuss related work in three key areas to capacity planning for vertical search engines: index organization in Section 2.1, workload characterization in Section 2.2, and performance modeling in Section 2.3.

2.1 Index Organization

Different strategies for distributing the index of a document collection among machines have been discussed in literature. Tomasic and Garcia-Molina [40] compare the performance impact on query processing of two basic and distinct options for storing the inverted lists, namely document partitioning and term partitioning. In document partitioning, documents are distributed among a set of index servers and each server stores a full index for its subset of the documents. In term partitioning, the index is distributed across the set of index servers and each server stores full index information for a subset of the terms. Simulation experiments attempt to determine under which conditions each index organization is better, how each index organization scales up to large systems and what is the impact of key parameters, such as seeking time of the storage device, load level, and number of keywords in a "boolean and" query.

Jeong and Omiecinski [21] consider the document partitioning and the term partitioning approaches to physically divide inverted indexes in a shared-everything multiprocessor machine with multiple disks. By simulation, they study the performance impact of these schemes on boolean query processing under a number of workloads where the term frequencies in the documents, the term frequencies in the queries, the number of disks, and the multiprogramming level vary. Ribeiro-Neto and Barbosa [31] study how the performance of disjunctive queries is affected by the index organization, the network capacity, and the disk transfer rates, using a simple analytical model coupled with a small simulator. MacFarlane et al. [24] and Badue et al. [4] investigate the performance impact on parallel query processing of the two distinct types of index organizations (either document or term partitioning), using a real case implementation. Sornil and Fox [37] and Badue et al. [2] compare the performance of hybrid partitioning against document partitioning and term partitioning. In the hybrid scheme they divide an inverted list into a number of equal sized chunks, which are randomly distributed to nodes in the system.

The previous comparisons of different strategies for distributing the index of a document collection have reached conclusions that are dependent on the use case considered. Whereas Tomasic and Garcia-Molina [40] and Sornil and Fox [21] find in favor of document partitioning, term partitioning has been pointed out as a better alternative by Ribeiro-Neto and Barbosa [31] and Badue et al. [4]. Further, Sornil and Fox [37] and Badue et al. [2] show that the hybrid partitioning outperforms both document partitioning and term partitioning. Regardless of the conclusion, these studies suffer from limitations such as the use of artificial or small sets of documents or queries, which are unlikely to be predictive of real-world behavior, and the use of (perhaps too simple and thus unrealistic) simulation models to estimate response time.

Under conditions better approximating those of a real-world large-scale search engine, Moffat et al. [26, 27] introduce a pipelined query evaluation methodology based on term partitioning, in which partially evaluated queries are passed amongst the set of index servers that host the query terms. Moffat et al. [26] examine methods for load balancing in the pipelined query evaluation methodology based on term partitioning and propose a suite of techniques for reducing net querying costs. In particular, they explore the load distribution behavior that pipelining displays, and show that the imbalances can be addressed by techniques that include predictive index list assignments to nodes and selective index list replication. An important conclusion of their experimental investigation is that document partitioning retains its leading position as the method against which others must be judged.

We thus adopt document partitioning for distributing the index of our document collection among index servers (see Section 3) because of its superior performance and popularity in large-scale search engines citeBarroso03,Risvik03.

2.2   Workload Characterization

Previous work on query characterization for Web search engines mainly focuses on the characterization of user search behavior and user search goals to enhance the relevance to users of the provided answers, i.e., to improve the search *efficacy*. Silverstein et al. [36] present an analysis of individual queries, query duplication, and query sections in a query log from the AltaVista Search Engine. Spink et al. [38] examine the query reformulation by users, and particularly the use of relevance feedback by users of the Excite Web search engine. Rose and Levinson [33] describe a framework for understanding the underlying goals of user searches. Baeza-Yates et al. [9] analyze query log data and show several models about how users search and how users use search engine results. Chau et al. [17] study the information needs and search behavior of the users of a Web site search engine and compare them with those of general-purpose search engine users. Kammenhuber et al. [22] use client-side logs to evaluate user behavior in what they call Web search click-streams, i.e., search-induced clicks on the answer page provided by the search engine and the subsequently visited hyperlinked pages.

Nevertheless, characterizing the workload imposed by queries on typical Web search engines is crucial not only for evaluating search *efficiency* in terms of, for instance, query response times, but also for designing predictive models. This is our focus in this paper. Beitzel et al. [12] analyze hourly variations in query traffic and remark that the number of queries issued is substantially lower during non-peak hours than peak hours. In our work, in addition to confirming this result for two different real-world vertical search engines, we characterize four key aspects of the workload imposed on these search engines, namely query and term popularity, number of terms per query, query interarrival times and query service times, with the goal of driving the design and evaluation of a performance model and its applicability to capacity planning (see Section 4).

2.3   Performance Modeling

Although many performance models exist for capacity planning of different systems citeMenasce04, the availability in the literature of performance models is rather limited for Web search engines in general and in particular for vertical search engines. Cacheda et al. [15] present a case study of different architectures for a distributed information retrieval system, in order to provide a guide to approximate the optimal architecture with a specific set of resources. Using a simulator based on an analytical model for query processing (similar to the one described in citeRibeiro-Neto98), they analyze the effectiveness of a distributed, replicated, and clustered architecture simulating a variable number of workstations. Nevertheless, their analytical model makes the simplifying assumption that service times are balanced if index servers manage a similar amount of data when processing a query. Chowdhury and Pass [19] introduce a framework based on queueing theory for analyzing and comparing architectures for search systems in terms of their operational requirements: throughput, response time, and utilization. Like in citeCacheda04, the proposed queueing model is not driven by observations from real workloads, also assuming a perfect balance among the service times of index servers that process an equal number of documents per query. Further, they do not verify the accuracy of their model by comparing its predictions with experimental results.

In contrast, we have previously found citeBadue07 that, even with a balanced distribution of the document collection among index servers, the heterogeneous use of disk cache in the homogeneous index servers leads to imbalances in query service times (see Section 3.4). Based on these previous findings, we here perform a detailed characterization of the workloads of two real-world vertical search engines, using their results to propose a performance model which, unlike previous work, captures the impact of this imbalance on the average query system response time.

3 An Architecture for Vertical Search Engines

3.1 Cluster of Index Servers

Modern search engines typically rely on computational clusters for query processing [13, 11, 32]. Such clusters are mostly composed of a single broker and p index servers. Figure 1 illustrates this architecture for a typical vertical search engine.
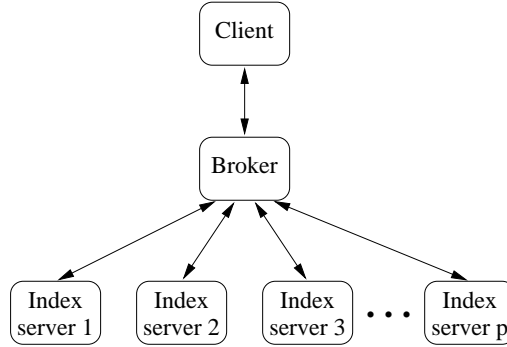

Figure 1: Architecture of a typical search engine.

The broker receives user queries from client nodes and forwards them to the index servers, triggering the parallel query processing through the p local subcollections. Each index server searches its own local subcollection and produces a partial ranked answer. These partial ranked answers are then sent to the broker where they are combined through an in-memory merging operation. The final list of ranked documents is then sent back to the user.

We note that the described architecture is for a single query processing cluster, which constitutes the basic unit of modern search engines. Large scale modern search engines basically replicate this cluster unit to support a higher query arrival rate [13, 11, 32].

3.2 Index Organization

An inverted index is adopted as the indexing structure for each subcollection. Inverted files are useful because they can be searched based mostly on the set of distinct words in all documents of the collection. They are simple data structures that perform well when the pattern to be searched for is formed by conjunctions and disjunctions of words [10, 45, 47].

The structure of our inverted indexes is composed of a *vocabulary* and a *set of inverted lists*. The vocabulary is the set of all unique terms (words) in the document collection. Each term in the vocabulary is associated with an inverted list that contains an entry for each document in which the term occurs. Each entry comprises a document identifier and the within-document frequency $f_{t,d}$ representing the number of occurrences of term t within the document d.

The documents of the whole collection are uniformly distributed among the index servers. We assign each document to an index server randomly, a policy that works well in balancing storage space utilization among servers [5, 3]. Let n be the size of the whole document collection. A uniform distribution of the documents among the p index servers leads to a size b of any local subcollection equal to $b = n/p$. That is, the size of any local inverted file is $O(n/p)$. This type of index organization, referred to as a *document partitioning*, is currently the de facto standard in all major search engines [11, 32].

3.3 Parallel Query Processing

In this paper, we use the standard vector space model [34] to rank the documents in the index servers. Modern search engines also adopt link-based ranking, such as PageRank [14], combined

with a complex text-based scoring function, to rank documents. However, since link information is pre-computed offline as a global measure, its usage has only limited impact on performance. Further, while we adopt here a simple ranking function, our model can capture the behavior of much more complex ranking functions, once key parameters are measured empirically.

In the vector model, queries and documents are represented as weighted vectors in a t-dimensional space, where t is the number of terms in the vocabulary of the collection. Each term-document pair is weighted by the frequency $tf_{i,j}$ of term k_i in document d_j and the inverse document frequency idf_i of the term k_i among the documents in the whole collection. The rank of a document with regard to a user query is computed as the cosine of the angle between the query and document vectors. Using the idf weight implies that global knowledge about the whole collection is available at the index servers. This can be accomplished if index servers exchange their local idf factors after the local index generation phase. Each index server may then derive the global idf factor from the set of local idf factors [31].

In our experiments, a client machine submits queries to the broker according to a query arrival distribution. This broker then broadcasts each query to all index servers. Once each index server receives a query, it retrieves the full inverted lists relative to the query terms, intersects these lists to produce the set of documents that contains all query terms (i.e., the conjunction of the query terms[1]), computes a relevance score for each document, sorts them by decreasing score, and sends its ranked answer to the broker. Each query term k_i is processed by decreasing idf_i, i.e., by increasing order of the number n_i of documents containing the term k_i, thus leading to a significantly more efficient conjunction of their inverted lists. As soon as the ranking is computed, the top ranked documents at each index server are transferred to the broker machine. The broker is then responsible for combining the partial ranked answers received from the index servers through an in-memory merging operation. The final list of top ranked documents is then sent back to the client machine.

We note that search engines that deal with large document collections perform a partial evaluation of the inverted lists relative to query terms instead of a full one [29]. Nevertheless, in our experimental setup (see Sections 5 and 6), we do not introduce any bound on the number of entries of the inverted lists. By doing so, our performance model, parameterized from experimental measures, provides conservative estimates of system performance, which are more adequate for capacity planning purposes. Morever, for simplicity, we adopt a single processing thread at each index server. The use of multiple processing threads at index servers is left for future work.

3.4 Imbalance in Query Service Times among Homogeneous Index Servers

In the architecture for parallel query processing, characterized by a local partitioning of the document collection, the response time of a query is determined by the service time of the slowest index server. As a consequence, any imbalance in service times among index servers increases the response time of a query executed by the cluster of servers. Therefore, it is critically important to avoid imbalance among index servers if higher performance is to be achieved.

A common counter-measure against imbalance is to distribute the whole collection of documents among homogeneous index servers (i.e., servers with identical hardware and software configuration) in a balanced way, such that each server handles a similar amount of data for processing any given query. At first glance, as a consequence of having similar data volumes handled at each server for a given query, one would expect that service times at the homogeneous index servers would also be approximately balanced. Indeed, this idealized scenario of balanced service times is a usual assumption taken by the few previous theoretical models for Web search engines available in the literature [15, 19].

However, we have previously shown that, in spite of the homogenous configuration and balanced amount of processing data, a non-negligible imbalance in per-query service time may arise within a cluster of homogeneous index servers [3]. The key reason for this imbalance, which can be quite significant in certain scenarios, is the heterogeneous use of disk cache at the different index servers.

---

[1]Taking the conjunction of the query terms is now standard practice on modern search engines.

In other words, for a given query q, even though all participating index servers tend to retrieve a similar amount of data, some index servers may experience much shorter service times for q if the needed documents are found in the disk cache maintained in *main memory* by the operating system. In contrast, the remaining index servers, having to retrieve the documents from the hard disk, will experience much longer transfer delays, thus leading to an overall imbalance in service times.

In [3], we show that the heterogeneous disk cache behavior is directly affected by correlations between two key workload aspects, namely, non-uniform term popularity distribution and sizes of the corresponding inverted lists. We also show that the main memory size at the homogeneous index servers, which defines the availability of resources for their disk caches, and the number of index servers are two other factors that impact disk cache behavior, and thus indirectly lead to per-query service time imbalance. In particular, the smaller the amount of main memory, the greater the chance that, while processing a query, some index servers might access their disk caches while the others have to access the hard disk. Moreover, for a fixed amount of main memory, the larger the number of participating index servers, the higher the chance that some of them will find documents in the disk cache. In both cases, the imbalance in service times increases.

4   Workload Characterization

This section presents the first step of our capacity planning methodology, namely, the characterization of real search engine workloads. The primary goal of this characterization is to build knowledge about key properties of real workloads and to draw insights into the design of cost-effective performance models. By cost-effective, we mean a model that is simple and yet accurate enough for practical capacity planning decisions in real setups. Additionally, we also provide models that best describe different workload characteristics that can be used in the design of realistic workload generators, although addressing this particular research venue is outside the scope of the current paper.

Our characterization relies on past access logs containing queries to two vertical search engines no longer operational, namely TodoBR[2] and Radix[3], both focused on the Brazilian Web. Additionally, we also have access to the workload of two much larger Web search engines no longer operational, namely AllTheWeb[4] and AltaVista[5], both focused on the whole Web. We note that the current availability of query datasets from modern operational search engines is rather restricted because such data are usually considered sensitive by search engine operators. Thus, the access to *four* different query workloads, from four different real search engines, sets our work in a priviledged position compared to most previous related efforts.

Table 1 presents an overview of our datasets along several dimensions including total period covered by each log, total number of queries as well as numbers of unique queries and unique terms. The two general search engines present much heavier query loads than TodoBR and Radix, which is expected, since the former are worldwide search engines and the latter regional ones. In particular both AllTheWeb and Altavista have average daily loads that are two orders of magnitude heavier than the typical loads experienced by TodoBR and Radix.

The following subsections present our characterization of five relevant workload aspects, namely, query length, query popularity and term popularity, query interarrival times and query service times. The design and validation of our performance model, driven by our characterization findings, is presented in Section 5.

---

[2]TodoBr (www.todobr.com.br) is a trademark of Akwan Information Technologies, which was acquired by Google Inc. in July 2005.

[3]Radix (www.radix.com.br) stopped its operation in 2001.

[4]AllTheWeb (`www.alltheweb.com`) is a trademark of Fast Search & Transfer company, which was acquired by Overture Inc. in February 2003. In March 2004 Overture itself was taken over by Yahoo!

[5]AltaVista (`www.altavista.com`) was bought in 2003 by Overture Inc., which was acquired by Yahoo! in the same period.

Table 1: Overview of our query datasets for vertical and general search engines.

	Vertical		General	
	TodoBR	Radix	AllTheWeb	Altavista
Start Date	Jan 01 2003	Jan 01 2003	Sep 01 2003	Sep 28 2001
End Date	Aug 31 2003	Aug 31 2003	Sep 21 2003	Oct 03 2001
Number of days	243	243	21	6
Number of queries	6,806,795	19,934,196	25,080,586	7,169,365
Number of unique queries	1,552,735	2,830,854	6,902,160	2,096,598
Number of unique terms	228,396	358,406	4,408,672	820,817
Avg. number of queries per day	28,012	82,034	1,194,314	1,194,893

4.1 Query Collection

We start by characterizing the query collection provided by each of the two vertical search engine datasets. In particular, three important aspects that describe each collection are query length (i.e., number of terms per query), query popularity and term popularity.

The query length, i.e., the number of terms per query, directly impacts the processing demand imposed for document retrieval at the index servers. We found a median query length equal to 2 for both TodoBR and Radix datasets, whereas the average query length is 2.02 and 1.91, respectively. These results suggest a significant trend towards queries composed of just one to a few terms. In fact, Table 2, which shows the query length distributions, confirms the prevalence of very short queries—i.e., queries containing at most two terms—for the two datasets.

Table 2: Distribution of query lengths in the vertical search engine datasets.

Number of terms	TodoBR	Radix
1	0.32	0.35
2	0.41	0.43
≥ 3	0.27	0.22

Next, we analyze the diversity of queries and of terms in queries. Figures 2(a) and 2(b) show the distributions of popularity (i.e., frequency) of unique queries and of unique terms in the TodoBR and Radix datasets, respectively. Although the query datasets cover different query loads, the distributions of popularity of queries and of terms in queries are quite similar. All of them present strong skews towards the most popular elements, while still exhibiting heavy tails. In fact, all of them follow Zipf distributions, as in Baeza- Yates et al. [6, 7]. That is, given that $Prob(E_n)$ is the probability that the nth most popular element (query or term) occurs in the dataset, then $Prob(E_n) \propto n^{-\alpha}$, where α is an empirical parameter. By fitting a straight line to each curve in the log-log plots presented in Figures 2(a) and 2(b), we estimate the value of the parameter α to be equal to 0.82 and 0.89 for the distributions of query popularity in the TodoBR and Radix datasets, respectively. Similarly, we found values of α equal to 0.98 (TodoBR) and 1.09 (Radix) for the distributions of term popularity. To illustrate the skew in these distributions, we verified that 1% of the queries accounts for 41% and 59% of the requests in the TodoBR and Radix datasets, respectively. The skews observed in the term popularity distributions are even stronger.

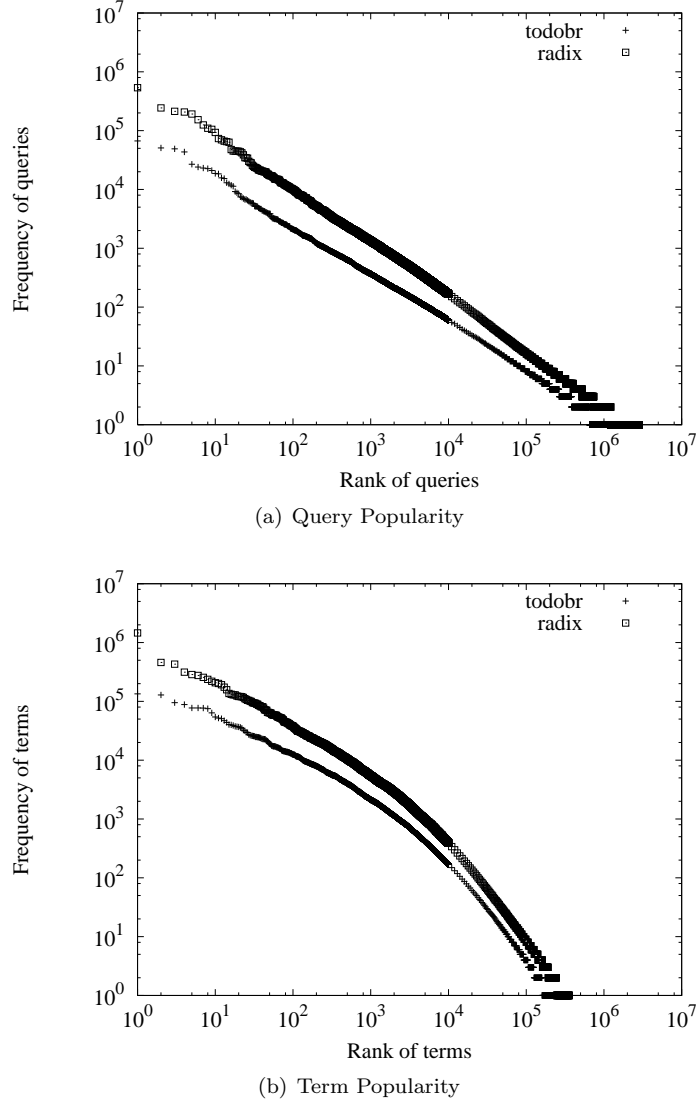(a) Query Popularity

(b) Term Popularity

Figure 2: Distributions of frequency of unique queries and frequency of unique terms in the vertical search engine datasets, plotted in log-log scale.

In practice, these skewed distributions of query popularity favor the use of disk caching at the index servers. The prevalence of a few terms per query combined with the strong skew in the term popularity distributions have similar implication. Ultimately, the heterogeneous use of disk cache at the index servers, exacerbated by a large number of servers or by a small amount of main memory, leads to significant service time imbalances, as discussed in Section 3.4. These imbalances must be taken into account when estimating the performance of a search engine, at the cost of greatly underestimating its capacity needs, as previous work does (see Section 2.3). Our strategy to capture service time imbalance in a performance model is described in Section 5.2.2.

4.2 Query Interarrival Times

One key performance characteristic of the workload is the query arrival process, characterized by the distribution of query interarrival times. This distribution must be characterized in periods during which the arrival process is stable (e.g., fixed arrival rate). Otherwise, the aggregation

of multiple workload behaviors might lead to unrepresentative distributions. Thus, our first step towards characterizing interarrival times consists of analyzing the temporal evolution of query arrivals.

A periodic behavior in the number of query arrivals is observed in the two datasets, in different time scales. To illustrate this behavior, Figure 3 presents the number of query arrivals within 60-minute bins over the whole duration of the TodoBR dataset. The curve for Radix shows similar qualitative profiles, being thus omitted. Figure 4 shows the number of query arrivals per 60-minute bin, for each day of the week, averaged over all weeks, for the two datasets. The query load in both datasets presents clear hourly load variations, with load peaks during the day and drops during the night. There are also significant load changes from working days and the weekend. Whereas in TodoBR, the load decreases during weekends, Radix experiences an opposite trend. These results are summarized in Table 3, which shows the average query arrival rate (in queries per second) over all periods covered by each dataset.
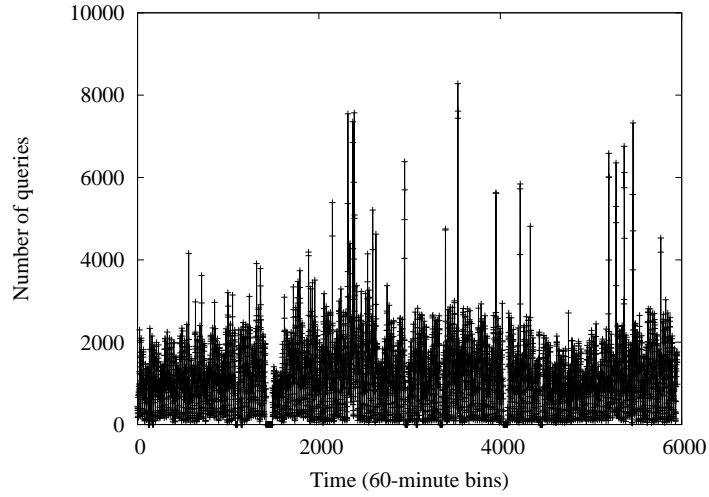

Figure 3: Query workload variation in the TodoBR dataset.

Table 3: Average query arrival rate (queries/second) in the vertical search engine datasets.

Day	Dataset		
	TodoBR	Radix	Folded TodoBR
Sunday	0.48	1.88	16.27
Monday	0.69	1.36	23.58
Tuesday	0.70	1.45	23.79
Wednesday	0.67	1.40	22.77
Thursday	0.70	1.40	23.80
Friday	0.61	1.33	20.77
Saturday	0.47	1.80	16.05

Before proceeding, we make an important observation that drives our next steps. In spite of the number of available query datasets being rather limited, it is even harder to obtain datasets for the collection of documents used by Web search engines to generate an answer page for each received query. This is because the set of collected documents is seen as strategic and proprietary information by the major Web search operators. A similar trend characterizes collections of documents found in vertical search engines. However, access to the document collection is of
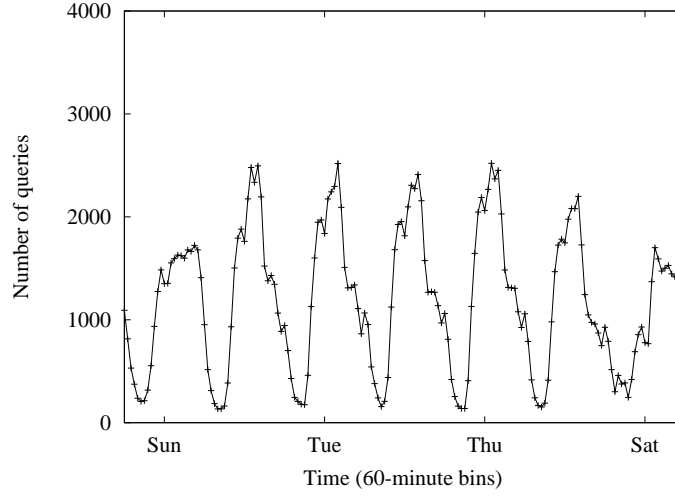
(a) TodoBR
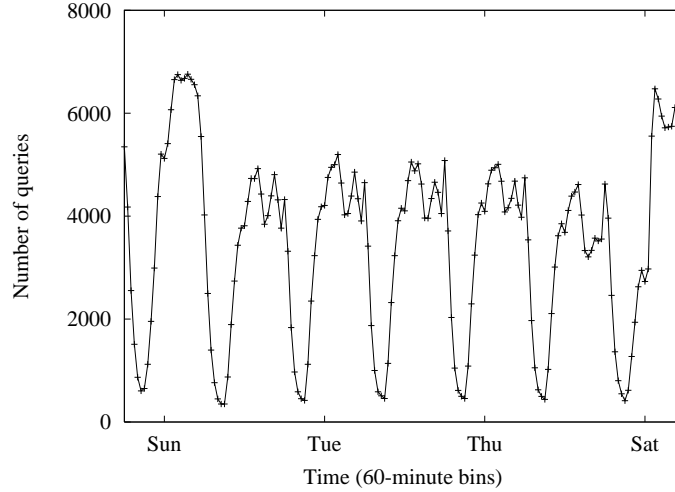
(b) Radix

Figure 4: Average number of queries over time (modulo one week) in the vertical search engine datasets.

fundamental importance to characterize query service times using real experiments, as we do in the next section, as well as to the experimental validation of our performance model, presented in Section 5.3.

Although we have access to four different query datasets (including those from two general search engines), we only have access to the document collection of one of them, namely the vertical search engine TodoBR. Restricting our experiments to the TodoBR dataset only would be constraining since, as shown in Figure 4 and Table 3, this search engine is somewhat light-loaded in terms of query arrival rates as compared to Radix. We note, however, that, though relatively light-loaded, the TodoBR dataset has characteristics (distribution of the number of terms per query, skew in query popularity and term popularity, and daily/hourly load variations) that are qualitatively similar to those found in the other datasets, including those of general search engines, being thus representative of them in terms of these workload aspects.

Thus, in other to obtain a complete (query and document) dataset that is representative in

terms of the key performance characteristics of other real datasets but has a heavier load, we apply a folding procedure to the TodoBR dataset. This procedure boosts the query arrival rate while still preserving the distribution models that best represent the performance characteristics under analysis. It consists of breaking the datasets into a number of consecutive equal-size time windows (e.g., 1 day, 1 week, 4 weeks, etc) and merging all queries falling into corresponding periods of all time windows. As an example, one could fold the dataset by considering a time window of 1 week and merging all queries that arrive on each day of the week (e.g., Monday), using data from the entire dataset. The shorter the time window considered, the larger the load boosting factor. In this paper, we fold the original TodoBR dataset by considering a time window equal to 1 week, although other windows could have been used leading to similar (qualitative) results. Figure 5 presents the daily load variation in the resulting folded TodoBR dataset, whereas the far right column in Table 3 shows the average query arrival rates. The folded TodoBR dataset will be used, together with the corresponding document collection, in the experiments discussed in Sections 4.3 and 5.3.
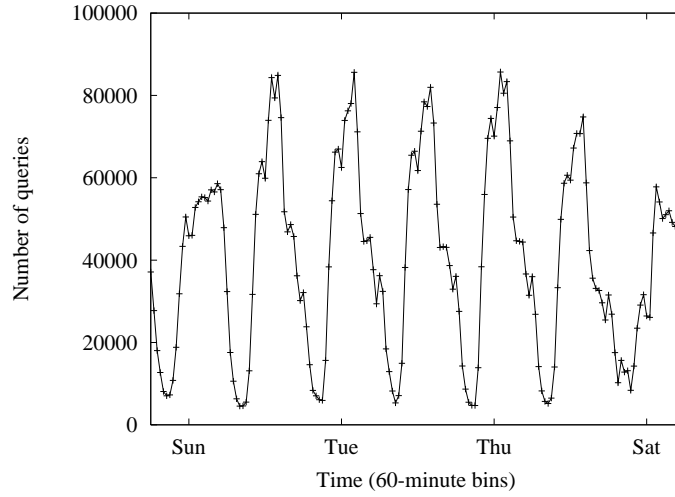

Figure 5: Number of queries over time in the folded TodoBR dataset.

We finally turn to the characterization of the query interarrival times. Given the high hourly and daily load variations observed in the original TodoBR and Radix datasets as well as in the folded TodoBR dataset, we characterize query interarrival times separately for a number of different heavy-loaded one-hour periods, during which arrival rates are stable. We evaluate the fittings provided by a set of well-known distribution models to the interarrival time distribution measured in each such period. The distribution models considered are Exponential, Gamma, Weibull, Lognormal and Pareto. The fitting of each distribution is evaluated by computing the sum of the squares of the differences and the maximum difference (also known as Komogorov-Smirnov goodness of fit test [16]) between the measured interarrival time distribution and the best-fit provided by each of these well-known distributions.

For all time periods analyzed, in all three datasets, we found that the Exponential distribution[6] presents a fairly reasonable fitting compared to the Gamma and Weibull distributions, whereas the Lognormal and Pareto distributions fail to model the observed data accurately. In order to illustrate how close the fitting provided by the Exponential distribution is to the measured interarrival time distribution, in comparison with the alternative distributions, we show the curves of measured query interarrival times as well as the best-fits provided by each distribution in one selected high-load hour of the folded TodoBR dataset in Figure 6.

---

[6]The probability density function (PDF) of the Exponential distribution is given by $f(x) = \frac{1}{\mu} e^{-\frac{1}{\mu} x}$, where μ is
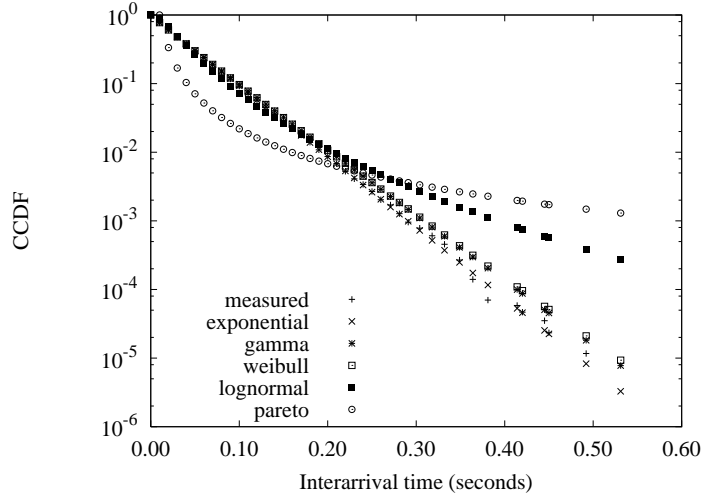
Figure 6: Distribution of query interarrival times in the folded TodoBR dataset (one 1-hour period): measured and predicted by various distribution models.

4.3 Query Service Times

To characterize query service times, we set up an experimental search engine, according to the architecture described in Section 3, in a dedicated environment consisting of a cluster of 8 homogeneous index servers and a single broker. We used a test collection composed of roughly 10 million Web pages collected by the TodoBR search engine from the Brazilian Web in 2003. The collection was uniformly distributed over the 8 index servers, resulting in a local subcollection of size $b = 1.25$ million pages. Each index server executes a simple ranking computation that combines text scoring with link analysis. We also instrumented the index servers to collect and dump into a file the total time spent servicing each query.

We ran a series of experiments issuing queries extracted from the folded TodoBR dataset. For each issued query, we collected the service time measured at each index server. In particular, the following discussion refers to the service times collected for a set of $85,604$ queries, extracted from a high-load one-hour period of that dataset, and measured after warming up the index servers.

We evaluate the fitting provided by well-known distribution models to the service time distribution measured in *each* index server. As for the query interarrival times, we considered the fittings provided by Exponential, Gamma, Weibull, Lognormal and Pareto distributions, and evaluated them using the same criteria discussed in Section 4.2. We found that Exponential, Gamma, Weibull and Lognormal distributions provided fairly reasonable fittings, whereas the Pareto distribution fails to model the observed data accurately. Figure 7 illustrates the fittings of each distribution model to the service times measured at one index server. Among the four best fittings, we choose to model query service times at each index server using an Exponential distribution. This decision is based on two main reasons. Firstly, our results demonstrate that the Exponential distribution approximates the observed service times per index server within fairly reasonable bounds. In fact, Figure 7 shows that noticeable deviations emerge only for service times greater than 0.12 seconds, accounting for a very small percentage (3%) of the queries. Secondly, the Exponential distribution is much simpler to model than the other ones, depending only on one parameter (μ) given by the average service time. Moreover, by comparing the fittings for all index servers, we found that the *same* Exponential distribution (i.e., same average service time and thus same distribution parameter μ) fits reasonably the service times measured in *all* index servers.

Therefore, from the characterization of our two vertical search engine workloads, we learned
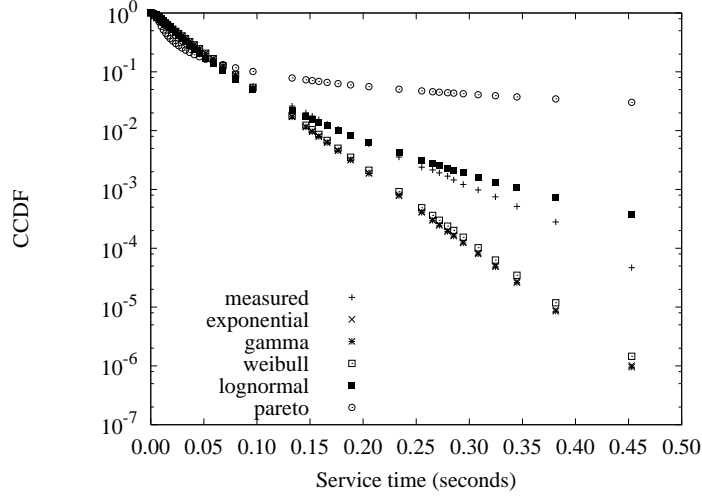
the distribution mean.

13

Figure 7: Distribution of query service times at one index server of the (folded) TodoBR dataset: measured and predicted by various distribution models.

that the skewed query and term popularity distributions combined with typically small query lengths may ultimately lead to significant service time imbalances among homogeneous index servers, and that query interarrival times and service times can be well approximated by Exponential distributions. Even more, query service times at different index servers follow the same Exponential distribution. These findings will drive our performance modeling strategy, which will be discussed in Section 5.

5   Performance Model for Vertical Search Engines

This section introduces the second step of our capacity planning methodology, namely, the performance model for vertical search engine architectures (described in Section 3). Our goal is to have a model that can answer questions such as:

(i) Given a collection composed of n documents distributed over p machines, what kind of average query response time guarantees can one expect?

(ii) What kind of optimization in machine resources might yield a reduction in the average query response time to meet a service level objective defined by the manager of a vertical search engine?

(iii) What is the minimum number of replications of the cluster of index servers that will guarantee that, on average, the query response time in a peak period will not exceed the threshold defined by the manager of a vertical search engine?

Our capacity planning strategy relies on a queueing-based analytical model to estimate the *average query system response time*. Its design was driven by the empirical observation that a tool to be useful to operators of a vertical search engine should be easy to configure and to apply in practical scenarios. Thus, model simplicity is of the utmost importance, even if it comes at the cost of a *reasonable* compromise in model accuracy. As will be shown, our model is simple, relies on easy-to-collect data, and still has reasonable accuracy in providing conservative estimates for capacity planning.
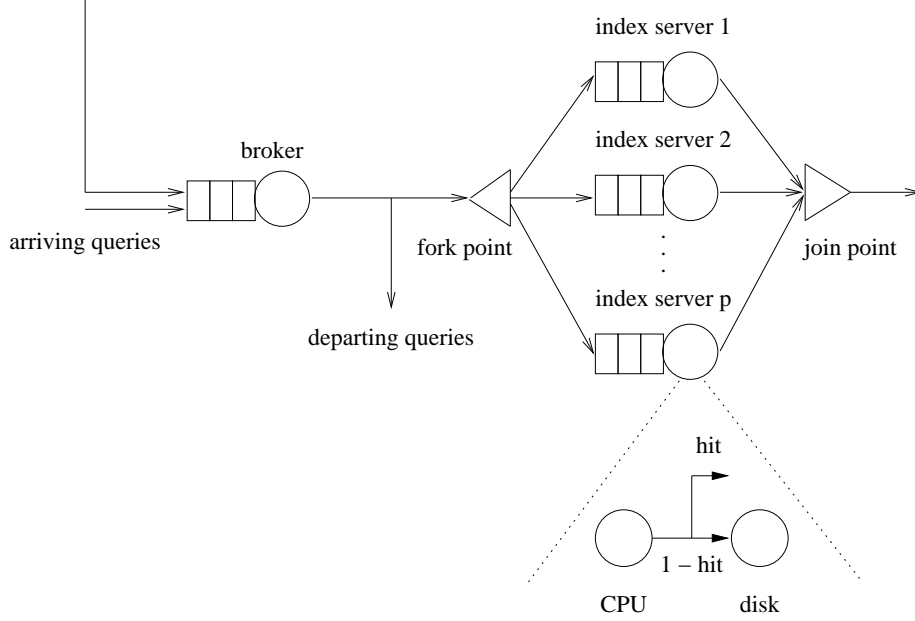
14

Figure 8: Queueing network for a vertical search engine.

5.1 Model Overview

The vertical search engine is represented by the queueing network described in Figure 8. Before describing its main components, we introduce some basic definitions that are useful to understand:

Definition 1: The *service time* (or demand) of a query at a resource (e.g., a server) is defined as the time the query spends receiving service at the resource.

Definition 2: The *residence time* of a query at a resource (e.g., a server) is defined as the sum of time the query spends waiting in the queue to be served (i.e., *waiting time*) and its service time.

Definition 3: A *fork-join queueing network* [25] composed of a number of service centers (e.g., index servers) is used to model parallelism and concurrency in computer systems. When starting a concurrent processing stage, a task is split (i.e., *fork*) into a number of identical subtasks. Each subtask is then sent to a service center, where it is executed independently from the other subtasks. When a subtask finishes execution, it waits until all its sibling subtasks finish (i.e., *join*). Only at this moment, does the task complete execution and leave the network.

As represented in Figure 8, we model a vertical search engine as an open queueing network composed of the broker and the subsystem of index servers. We assume the index servers have homogeneous resources (as would be the case in several real scenarios) and that the collection of documents is uniformly distributed over all servers. Based on the results presented in Section 4, we also assume that query service times at index servers are exponentially distributed. Finally, the network connecting index servers and broker is typically a high-speed network and, as observed in our experiments, introduces negligible delays to the query system response time. Therefore, it is not explicitly represented in our model.

In our search model, the index server subsystem is modeled as a fork-join queueing network composed of p index servers. The queueing discipline at each index server is FCFS (First-Come First-Served). The behavior of this fork-join network, in which each task represents an arriving

query and each service center models an index server, mimics the parallelism in processing queries by the index servers and the synchronization introduced at the broker for combining partial results. Mean Value Analysis (MVA) [30] offers an efficient solution for product-form queueing networks. In particular, MVA can be used to produce performance estimates for each individual index server. However, the fork-join feature violates the assumptions required by the exact MVA solution. Thus, we use approximate MVA and bounding techniques [25] to solve the complete search model.

To process a query, an index server needs to retrieve the inverted lists related to the query terms from disk. Thus, query service time at the index server is dominated by disk time and, possibly, CPU time. However, due to the locality of reference of terms in the query log, an index server might find some or all of the inverted lists corresponding to the query terms in the disk cache (i.e., in main memory). Thus, some queries may not retrieve any data from disk at all. In fact, during our validation experiments (see Section 5.3), we found a non-negligible number of such queries in our workloads.

To capture the impact of disk cache, we refine our index server model as follows. We separately model the average demands for CPU and disk (average CPU and disk times, respectively), as well as the probability of full disk cache hit (i.e., *all* inverted lists for a query are found in the disk cache) at an index server. Note that the impact of partial disk cache hits is indirectly captured by the CPU and disk times. We also assume that queries may have different CPU times depending on whether they retrieve any data from the disk. Given that queries are processed sequentially by each index server (see Section 3), there is no queueing at any resource (CPU nor disk) of an index server.

The query residence time at the broker, which depends on the number of servers, consists of local processing for broadcasting the query to all index servers, receiving partial results from all servers, and merging the received partial results. This residence time at the broker is relatively low compared to the average query system response time, due to two basic reasons. First, broker's operation is fully carried out using main memory, thus demanding only CPU time as opposed to an index server's operation that is composed of CPU and disk demands. Second, all the tasks the broker executes are relatively simple tasks that do not take much CPU time. It should be noted that the broker does not have to make ranking computations and does not have to execute algebraic operations, other than comparing document ranks.

Table 4 presents the system and workload input parameters as well as the output parameters of our model.

5.2 Model Solution

This section describes our solution to the model in Figure 8, which we use to estimate the average query system response time of a vertical search engine. Our main design goals are simplicity and reasonable accuracy, while providing a conservative estimate on performance. Moreover, we are particularly interested in solutions that deliver a good tradeoff for heavy load scenarios, when the search engine is approaching saturation (i.e., server resource utilization close to 100%).

5.2.1 Index Server Model

This section derives the average query residence time at one index server. Since we assume no queueing at any resource of an index server, a conservative assumption, we start by introducing an abstraction for each server as a single service center. As a result, the average service time of a query at one index server is given by

$$S_{server} = hit S_{hit} + (1 - hit)(S_{miss} + S_{disk}). \tag{1}$$

Applying Little's result to an MVA equation for open networks [25], we estimate the average residence time of a query at an index server as:

$$R_{server} = \frac{S_{server}}{1 - \lambda S_{server}}. \tag{2}$$

16

Table 4: Input and output parameters of our model.

Inputs	Description
p	Number of index servers
λ	Query arrival rate
S_{broker}	Average service time of a query at the broker for a cluster with p index servers
S_{hit}	Average CPU time at an index server for a query that finds all inverted lists for the query terms in the disk cache
S_{miss}	Average CPU time at an index server for a query that retrieves data from disk
S_{disk}	Average disk time of a query at an index server
hit	Probability of a query finding all inverted lists for the query terms in the disk cache

Outputs	Description
R	Average system response time of a query
$R_{cluster}$	Average residence time of a query at the index server subsystem
R_{broker}	Average residence time of a query at the broker for a cluster with p index servers
R_{server}	Average residence time of a query at an index server
S_{server}	Average service time of a query at an index server
U_{server}	Total resource utilization of a query at an index server

We can also estimate the aggregated utilization of an index server resources by the query:

$$U_{server} = \lambda S_{server} \tag{3}$$

Similarly, the average query residence time at the broker can be also easily estimated using MVA:

$$R_{broker} = \frac{S_{broker}}{1 - \lambda S_{broker}}. \tag{4}$$

Finally, the average query system response time R is computed as the sum of the average query residence times at the broker and at the index server subsystem, i.e.:

$$R_{broker} + R_{server} \leq R, \tag{5}$$

where the inequality reflects the fact that R_{server} considers residence time in a single index server.

5.2.2 System Model

Recall that the index server subsystem is modeled as a fork-join network. There is no known closed-form solution for fork-join networks with more than 2 queues. Hence, the performance metrics of such networks must be computed using approximation and bounding techniques. An easy lower bound on the average query residence time in the fork-join subsystem is obtained by ignoring the synchronization delays and considering the average query residence time in the fork-join subsystem equal to the average query residence time at an index server (see Equation 2). Indeed, this is the solution adopted by Chowdhury and Pass [19] to estimate the average query system response time of a search system that, similarly to our work, is modeled as an open queueing network composed of a broker and a subsystem of index servers. However, as the number of index servers increases, we expect a significant deviation from this lower bound due to the synchronization overhead, a result of the imbalance among the index servers, as discussed in Section 3.4.

A number of approximations for queueing models with fork-join synchronization, with various degrees of complexity and accuracy, are available in the literature [20, 28, 1, 44, 39, 46, 43, 41, 23, 18, 42]. Nelson and Tantawi [28] propose a very simple upper bound on the average response time for fork-join queueing networks with exponential interarrival times and exponential service times (as verified in Sections 4.2 and 4.3, respectively), which depends only on the number of index servers p, and on the average query residence time at *one* server. Given the pth harmonic number $H_p = 1 + 1/2 + 1/3 + \cdots + 1/p$, the upper bound on the average query residence time at the index server subsystem is given by:

$$R_{cluster} \leq H_p R_{server}. \tag{6}$$

The harmonic number H_p is greater than 1 and provides an approximation to the fact that the residence time in the cluster is determined by the slower server. As will be shown in the following, we found this upper bound—although quite simple—yields reasonably accurate estimates of the average query residence time at the index server subsystem. Combining equations 2, 4 and 6, we obtain the following bounds on the average query system response time for our search engine:

$$R_{server} + R_{broker} \leq R \leq H_p R_{server} + R_{broker}. \tag{7}$$

We note that for a large number p of index servers, the pth harmonic number H_p converges to $\ln(p) + O(1)$, indicating that there is a logarithmic factor between the upper and lower bounds.

5.3 Model Validation

A series of validation experiments were executed in a dedicated environment consisting of a cluster of 8 homogeneous index servers and a single broker, with loads in the range of 10 to 50 queries per second. Each index server runs on a 2.4 gigahertz Pentium IV processor with 256 megabytes of main memory and a 120 gigabytes ATA IDE hard disk. The broker is an ATHLON XP with a 2.2 gigahertz processor and 1 gigabyte of main memory. All of them run the Debian Linux operating system kernel version 2.6. Index servers and broker are connected by a 100 megabits/second high-speed network. We used a test collection composed of roughly 10 million Web pages collected by the TodoBR search engine from the Brazilian Web in 2003. The inverted index for the whole collection occupies roughly 12 gigabytes. The collection was uniformly distributed over the 8 index servers, resulting in a local subcollection of size $b = 1.25$ million pages. Each index server executes a simple ranking computation that combines text scoring with link analysis. More sophisticated ranking computations can also be represented, given that our model parameters do not depend on specifics of the ranking computation. The query dataset used in our tests is composed of $85,604$ queries in a high-load hour of the folded TodoBR dataset.

The values of the model input parameters were easily obtained by retrieving statistics collected by the Linux operating system and made available at the */proc* pseudo-filesystem, during the experiment. We collected the value of each parameter for each query in our test dataset, by averaging the results for all queries at the end of the experiment. CPU and disk times for a query are collected from the */proc/stat* pseudo-file. To estimate the fraction of queries that found all inverted lists for the query terms in the disk cache (*hit*), we monitored the total number of sectors successfully retrieved from disk by each query, available in the I/O statistic field of the */proc/diskstats* pseudo-file. Table 5 presents the model input parameter values obtained in our experiments. The CPU and disk times at the index servers and hit probabilities are averages for all index servers.

Figure 9 shows the average query residence time at an index server, averaged over all index servers, as a function of the total query arrival rate. The "estimated" curve represents the results obtained with Equation 2, whereas the "measured" curve contains the average query residence times measured in all 8 index servers. As shown, our model captures reasonably well the average performance of a typical index server. For an arrival rate of 28 queries/second, the average utilization of the disk, the bottleneck resource at the index servers, is already almost 79%, and the estimated aggregated utilization of the index server resources (U_{server}) approaches 92%. Thus, the index server is approaching saturation. For this load, the error introduced by our model is

Table 5: Model input parameter values.

Parameter	Value
p	($\leq$) 8 servers
b	1.25 million pages
$S_{broker}, p = 2$	0.33 ms
$S_{broker}, p = 4$	0.39 ms
$S_{broker}, p = 8$	0.52 ms
S_{hit}	9.20 ms
S_{miss}	10.04 ms
S_{disk}	28.08 ms
hit	0.17

only 23%, reasonably small for response time estimates, as suggested in Menasce et al. [25]. In the case of arrival rates higher than 28 queries/second, our model predicts that the server saturates (i.e., server utilization is higher than 100%), which is indeed confirmed by our experimental measurements.
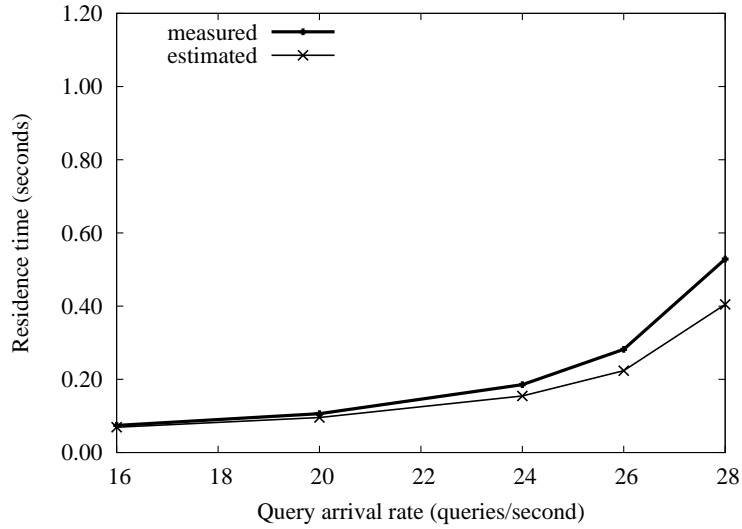

Figure 9: Average query residence time at an index server as a function of the query arrival rate ($p = 8$).

We now turn to the validation of the average query system response time. Figures 10 and 11 show experimental results as well as the lower and upper bounds on the average query system response time, estimated via Equation 7, as a function of the arrival rate and of the number of index servers, respectively. In Figure 11, in order to make a fair comparison, we kept the size of the subcollection b fixed (i.e., 1.25 million pages per index server), varying only the total number of servers p and, indirectly, the size of the total collection $n = pb$. The upper bounds on the average query system response time for a number of servers p equal to 2, 4, and 8 are 0.61, 0.84 and 1.10, respectively. This confirms the logarithmic factor between the upper and lower bounds, as indicated at the end of Section 5.2.2.

As the results in this subsection show, the lower bound on the average query system response time is a good approximation for systems with a small number of index servers and/or light loaded
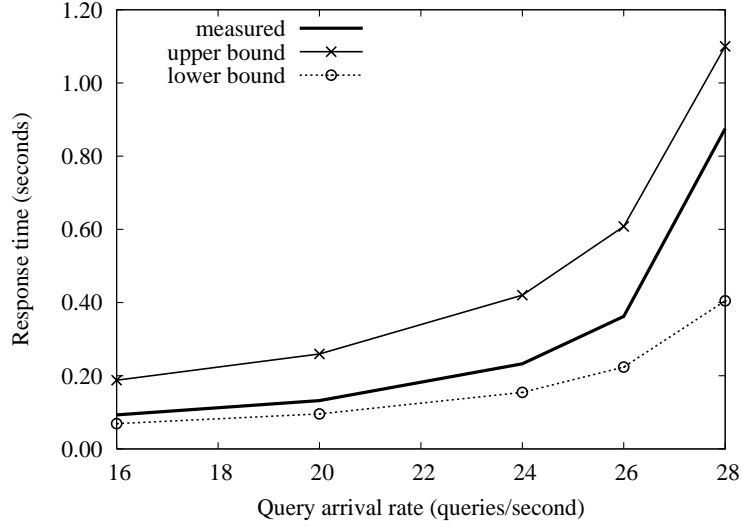
Figure 10: Average query system response time as a function of the query arrival rate ($p = 8$).
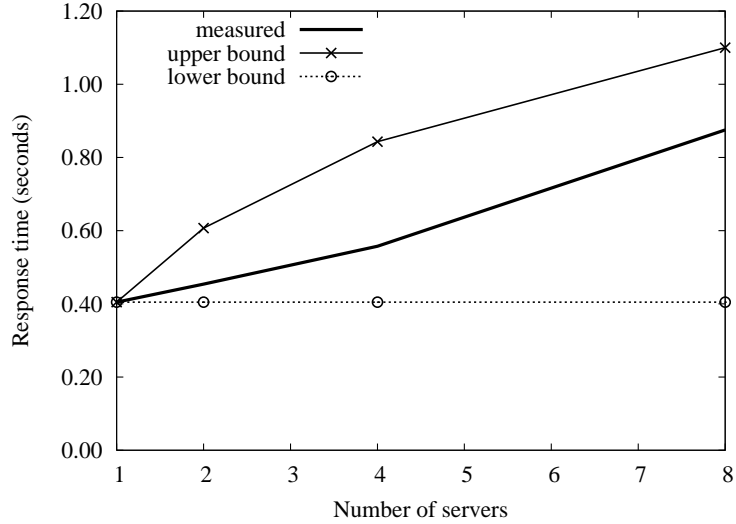

Figure 11: Average query system response time as a function of the number of index servers p ($\lambda = 28$ queries/second).

servers. However, as either the load or the number of index servers increase, the measured average query system response time deviates significantly from the lower bound due to the synchronization overhead. This contrasts with previous work that disregards imbalance in query service times among homogeneous index servers [19, 15]. In fact, we observe that the measured average query system response time approaches the upper bound for larger number of index servers and heavier loads. In particular, for $p = 8$ index servers and an arrival rate $\lambda = 28$ queries/second (i.e. close to saturation), the approximation error is only 20%. Therefore, the upper bound provides a simple-to-compute and yet reasonably accurate approximation of the average query system response time of heavy-loaded (and thus perhaps more realistic) vertical search engines. The achieved accuracy should be enough for cost-effective capacity planning decisions on such vertical search engines.

6 Performance Model Applicability

In this section, we discuss how to use our model for the capacity planning of vertical search engines. This is important because the number of index servers in a real cluster is usually large and maintaining large clusters operational is quite expensive. Furthermore, our model is easy to use and depends only on information that is usually available in both search logs and standard operating systems, as described in Section 5.1.

We consider "what if" questions, concerning future scenarios for a vertical search engine. We assume a collection of 1 billion pages. We also assume that each index server stores a subcollection of size $b = 10$ million pages—the largest collection we have available for experimentation, which requires 100 index servers to host the whole collection. To obtain the parameters of the index server model, we executed experiments in a reference system composed of a *single* index server running on a machine with configuration equivalent to the index server described in Section 5.3.

To analyze the impact of adopting a larger main memory on the query system response time, we also executed experiments in this index server with main memory four times larger than the reference machine. Note that we needed a single (upgraded) index server to collect the new parameter values, but once we have it we can predict performance with various numbers of servers. We used a test collection composed of roughly 10 million Web pages collected by the TodoBR search engine from the Brazilian Web in 2003, and a query dataset composed of $85,604$ queries in a high-load hour of the folded TodoBR dataset (see Section 4).

Based on these experiments, we determined parameter values for the index server model, such as the average CPU and disk times per query, and the probability of full disk cache hit at an index server for a query, while considering different main memory sizes. To obtain the average query service time at the broker as a function of 100 index servers (S_{broker}, $p = 100$), we fit a straight line to the values of S_{broker} ($p = 2, 4, 8$) estimated during our validation experiments (see Table 5). We found an accurate fitting (coefficient of determination of $R^2 = 9.999870 \times 10^{-1}$) given by $S_{broker} = 3.18 \times 10^{-2}p + 0.265$ milliseconds, which we used to derive $S_{broker} = 3.45$ milliseconds for $p = 100$. Table 6 presents the new parameter values used in our example.

Table 6: New model input parameter values used in our example.

Parameter	Value			
	Reference size for main memory	2x the reference main memory	3x the reference main memory	4x the reference main memory
p	100 servers			
b	10 million pages			
S_{broker}	3.45 ms			
S_{hit}	28.23 ms	33.38 ms	34.57 ms	34.68 ms
S_{miss}	35.31 ms	33.77 ms	32.66 ms	32.04 ms
S_{disk}	66.03 ms	35.89 ms	30.48 ms	26.14 ms
hit	0.02	0.09	0.15	0.18

Once the model parameters are computed, one can apply the model to derive the performance metrics of interest.

Case Study - The manager of a vertical search engine wants to guarantee that the average query system response time will not exceed 300 milliseconds. Further, by replicating clusters of servers the manager plans to support a combined arrival rate of 200 queries per second.

To estimate the behavior of the parameters involved in this case study, we apply our capacity

planning approach to the problem. Using the model with the parameters described in the second column of Table 6 (for an index server with the reference size for main memory), we calculate the upper-bound on the average query system response time as a function of the query arrival rate. The result, illustrated by the "baseline" curve in Figure 12, indicates that the baseline exceeds the threshold of 300 milliseconds per query established by the manager, even at very low query arrival rates.
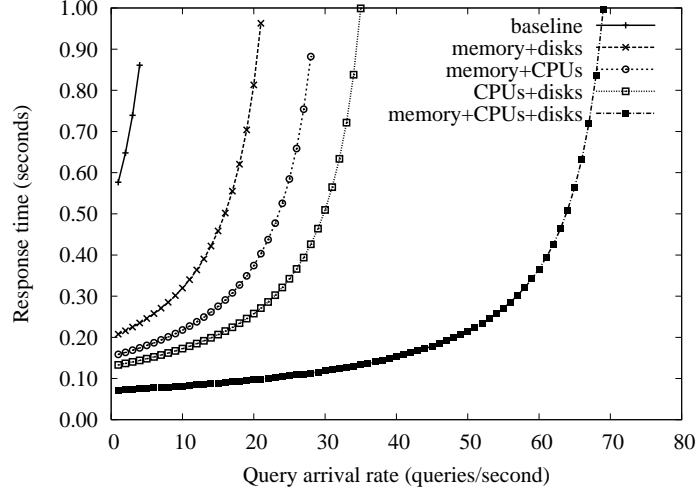

Figure 12: Upper bound on the average query system response time as a function of the query arrival rate derived in our example.

We want to evaluate what kind of optimization in the resources of index servers might yield a reduction in the average query system response time to less than 300 milliseconds. For this, we consider five scenarios. Also in Figure 12, the curves labeled "memory+disks", "memory+CPUs", "CPUs+disks" and "memory+CPUs+disks" represent the upper bound on the average query system response time for four scenarios where: (i) main memories are four times larger and disks are four times faster; (ii) main memories are four times larger and CPUs are four times faster; (iii) CPUs and disks are both four times faster; and (iv) main memories are four times larger, and CPUs and disks are both four times faster, respectively. Furthermore, we evaluate the impact of application-level caching on the response times predicted by our model based on recent experimental results found in the literature. These are the five scenarios analyzed in the following.

Scenario 1 - Main memories are four times larger and disks are four times faster.

In the first scenario, we want to evaluate the impact on query system response time of main memories that are four times larger and disks that are four times faster. This is reflected in the model parameter by dividing the disk time S_{disk} (described in the fifth column of Table 6 for an index server with main memory four times larger than the reference size) by a factor of four. Solving the model with the new parameters yields the new average response time upper bound. The results show that the upper bound on the average query system response time decreases significantly, with gains that reach approximately 4 times over the baseline system when it is approaching the point of saturation ($\lambda = 4$ queries/second). The reason for these gains is that the probability of a query finding all inverted lists in the disk cache increases with the size of the main memory, thus accelerating disk I/O. Indeed, when the main memory increases by a factor of four in our experiments, the probability of full disk cache hit increases by a factor of nine (0.18/0.02) and the demand for disks decreases by a factor of 2.53 (66.03/26.14), as indicated in Table 6. Nevertheless, the upper bound still exceeds the defined threshold even at light loads.

Scenario 2 - Main memories are four times larger and CPUs are four times faster.

In the second scenario, we want to assess the impact on query system response time of main memories that are four times larger and CPUs that are four times faster. The way of modeling the new CPUs is to divide the CPU times S_{hit} and S_{miss} (described in the fifth column of Table 6 for an index server with main memory four times larger than the reference size) by a factor of four. Using the model, we calculate the new average response time upper bound. The results indicate that the configuration with faster CPUs outperforms the configuration with faster disks, with gains that reach approximately 5 times over the baseline system when the latter is is approaching saturation ($\lambda = 4$ queries/second). There are two reasons for these gains. First, the demands for CPUs are larger than the demands for disks in the configuration with main memory four times larger than the reference size (see fifth column of Table 6). This implies that the optimization of CPUs yields a higher reduction in the average query system response time than the optimization of disks. Second, the optimization of CPUs improves both the performance of index servers and the performance of the broker, while the optimization of disks benefits only the index servers. Nevertheless, the upper bound still exceeds the defined threshold even at somewhat light loads ($\lambda \geq 17$ queries/second).

Scenario 3 - CPUs and disks are both four times faster.

In the third scenario, we want to verify the impact on query system response time of CPUs and disks that are both four times faster. For modeling the new resources, we divide the CPU and disk times (described in the second column of Table 6) by a factor of four. The solution of the model with the new parameters yields the new average response time upper bound. The results show that the configuration with faster CPUs and disks outperforms the configurations with larger main memories (in Scenarios 1 and 2), with gains of approximately 6 times over the baseline system when it is approaching the saturation point ($\lambda = 4$ queries/second). Nevertheless, the upper bound still exceeds the defined threshold at moderate to high loads ($\lambda \geq 22$ queries/second).

Scenario 4 - Main memories are four times larger, and CPUs and disks are both four times faster.

In the fourth scenario, we want to evaluate the impact on query system response time of main memories that are four times larger, and CPUs and disks that are both four times faster. This is reflected in the model parameters by dividing the CPU and disk times (described in the fifth column of Table 6 for an index server with main memory four times larger than the reference size) by a factor of four. The results indicate that the upper bound on the average query system response time is equal to 286 milliseconds at an arrival rate of 56 queries/second, which satisfies the service level objective for the search engine by bounding the average response time to less than 300 milliseconds per query. The results also show a remarkable reduction in the upper bound on the average query system response time, with gains that reach approximately 12 times over the baseline system when the latter is approaching its saturation point ($\lambda = 4$ queries/second).

We still have to meet the goal of supporting an arrival rate of 200 que-ries/second. For supporting a higher query arrival rate, the cluster of index servers is usually replicated [13, 11, 32]. Replication involves relatively small performance overheads, and approximately linear gains in the supported query arrival rate can be expected as a function of the number of mirrored systems. The objective of supporting an arrival rate of 200 queries/second can be achieved by creating 4 replicas of the cluster of 100 index servers, each replica supporting an arrival rate of 56 queries/second and guaranteeing a query system response time of 286 milliseconds. Therefore, our model indicates that a cluster composed of 400 index servers (4 cluster replicas × 100 index servers in a cluster) would achieve the desired performance of 200 queries/second. Some speculations suggest that large scale search engines may indeed adopt clusters with several thousands of machines, but, to the best of our knowledge, there is no publicly available data to support this information. If each index server handles a larger subcollection, the number of total servers in a cluster could be smaller. Thus, one must be cautious before extrapolating our illustrative results to an arbitrary cluster of any search engine, in particular large scale public Web search engines with a much larger scope than typical vertical search engines. In this case, for other document collections, for other types of machines, or for other information retrieval algorithms, the parameters of our model could still be estimated experimentally and applied following the proposed methodology.

Scenario 5 - Evaluating the impact of upgrade decisions.

We now use our model to analyze the impact on the overall performance of the system caused by alternative upgrade decisions, i.e. the influence on the overall performance of varying main memory sizes, CPU speeds, and disk speeds. Figures 13(a) to (d) show the upper bounds on the average query system response time as a function of main memory size, CPU speed and disk speed, for a fixed arrival rate (4 queries/second). The baselines for "disk speed" (i.e. disk speed equaling 1) and "CPU speed" (i.e. CPU speed equaling 1) in these figures correspond to the disk and CPU speeds for which the model parameter values are shown in Table 6. The results for different memory sizes are computed using the parameter values shown in columns $2-5$ in the same table.

(a) Reference main memory size

(b) 2x the reference main memory size

(c) 3x the reference main memory size
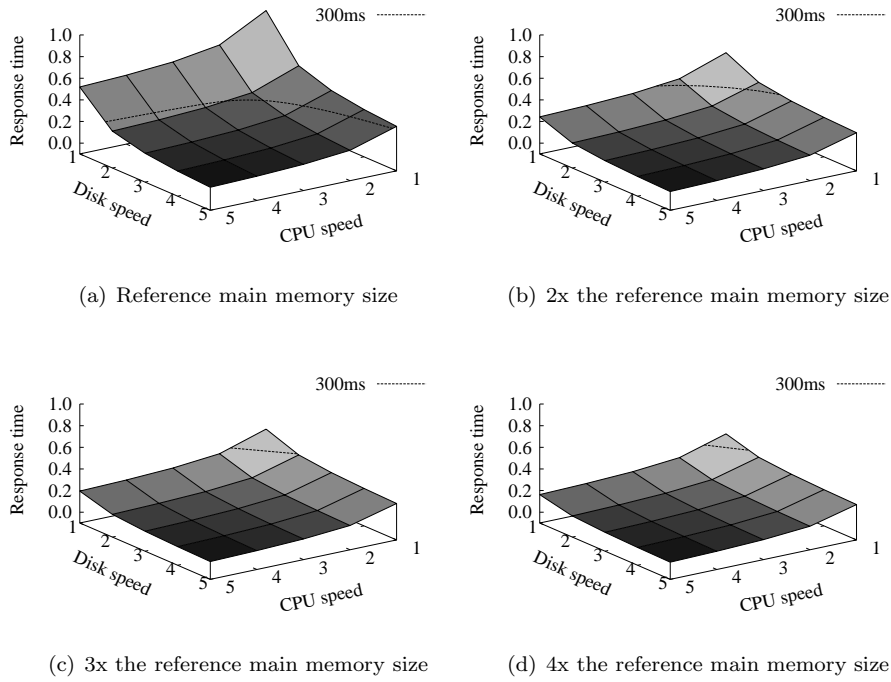
(d) 4x the reference main memory size

Figure 13: Upper bounds on the average query system response time (in seconds) as a function of main memory size, CPU speed (x times faster) and disk speed (y times faster), for a fixed arrival rate (4 queries/second).

We observe in Figure 13(a) that response time decreases more rapidly with the increase of disk speed than with the increase of CPU speed. This is a consequence of the much larger time for disk access to retrieve the inverted lists related to query terms, as compared to the case in which the lists are found in the disk cache. Since the available memory size is rather limited in this case, the demands for disk access are higher than demands for CPU processing, thus leading to increased response times. Nevertheless, as the main memory size increases, as shown in Figures 13(b) to (d), the system faces a contrasting trend: average response time decreases with the increase of CPU speed, decreasing at much slower rate with disk speed. When the main memory size is relatively large as compared to the size of the local index stored at the servers, there is more memory capacity available for the operating system to perform disk caching operations. In this way, CPU demands become higher than disk demands contributing to the decrease of response time. Note that results in this scenario consider only disk caching.

Scenario 6 - Influence of application-level caching of query results at the broker.

Although our model does not incorporate application-level caching of query results, we built upon recent results from the work presented in [8], in order to evaluate the influence on query system response time of caching query results at the broker. Baeza-Yates et al. [8] study the tradeoffs in designing caching systems for Web search engines, including caching of answers to a particular query. The strategy of caching query results is to keep in memory the list of documents associated with a given query. This scheme allows the broker to answer recently issued queries at a very low cost, since it is not necessary to process these queries.

Actually, based on the Equation 7, the upper bound on the average query system response time for our search system with caching of query results can be estimated as:

$$R \leq \left(H_p \frac{S_{server}}{1 - \lambda S_{server}} + R_{broker} \right)(1 - hit^{result}) + \frac{S_{broker}^{cache_{hit}}}{1 - \lambda S_{broker}^{cache_{hit}}} hit^{result} \qquad (8)$$

where $S_{broker}^{cache_{hit}}$ is the average service time at the broker for a query that finds its results in the application-level cache and hit^{result} is the probability of hit in the cache of query results at the broker.

(a) Reference main memory size

(b) 2x the reference main memory size

(c) 3x the reference main memory size
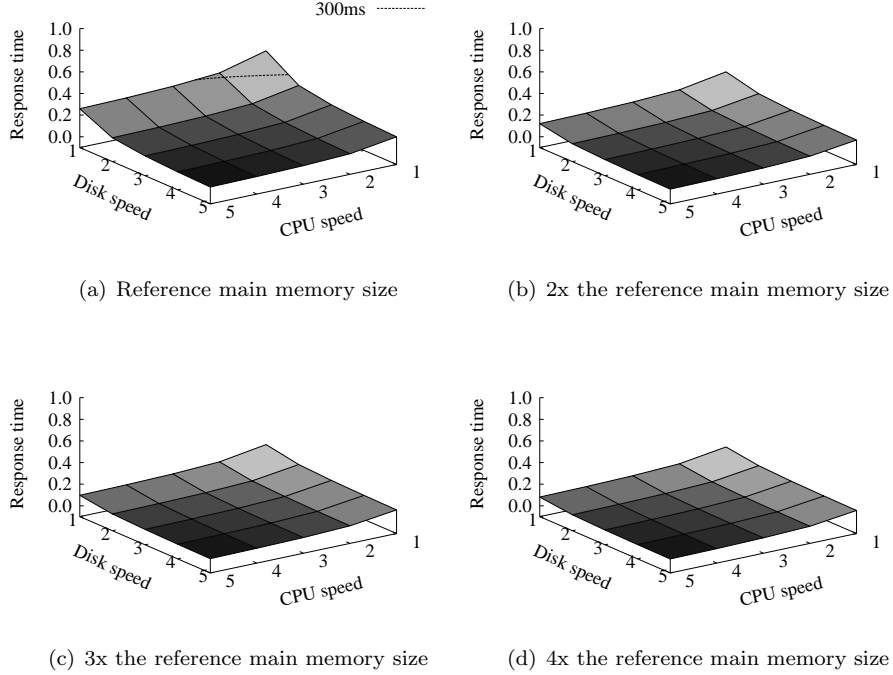
(d) 4x the reference main memory size

Figure 14: Upper bounds—that consider application-level cache of query results at the broker—on the average query system response time (in seconds) as a function of main memory size, CPU speed (x times faster) and disk speed (y times faster), for a fixed arrival rate (4 queries/second).

Using logs of one year of queries submitted to http://www.yahoo.co.uk from November 2005 to November 2006, Baeza-Yates et al. [8] estimated experimentally a cache hit ratio of 0.50, when considering a cache of query results with infinite memory, and an average time of 0.069 milliseconds to return a stored answer for a query. Applying these values to Equation 8 ($hit^{result} = 0.50$ and $S_{broker}^{cache_{hit}} = 0.069$ milliseconds), we calculate the new upper bounds on the average query system response time, which now consider application-level caching of query results at the broker as a function of main memory size, CPU speed and disk speed, for a fixed arrival rate (4 queries/second), as shown in Figures 14(a) to (d). We observe a significant decrease in response times through all system configurations as compared to the performance presented in Figures 13(a)

to (d), in which the scenario does not consider application-level caching. As a consequence, to support an arrival rate of 200 queries/second considering the same system configurations of Scenario 4, our model now indicates that a cluster composed of 300 index servers (3 cluster replicas × 100 index servers in a cluster, each replica supporting an arrival rate of 65 queries/second and guaranteeing an average query system response time of 282 milliseconds) would be able to achieve the desired performance, as it benefits from the performance improvements resulting from application-level caching.

7   Conclusions

As the main contribution of this paper, we proposed and investigated a methodology for analyzing the performance of search engines. Based on the proposed methodology, we developed a capacity planning model for vertical search engines that considers the imbalance in query service times among homogeneous index servers. Our model, based on queueing theory, is simple and reasonably accurate. To fine tune the model, we ran experiments on a small cluster of index servers. Once tuned, we compared the predictions of our model with results we measured empirically and found a high quality matching. Even at the saturation point, the predictions of our model were reasonably accurate.

We illustrated how to apply our model to predict average query system response time when adopting larger main memories, faster CPUs and disks than those in use, as well as applying an application-level caching of query results at the broker. We considered a realistic scenario, where a collection of 1 billion documents is distributed over 100 index servers. In this scenario, we showed that the manager of a vertical search engine can quickly reach predictions for upper bounds on the average query system response time without having to run any live experiments.

Given the complexity of maintaining large scale search engines, and the simplicity and reasonable accuracy of our model, we believe our model can be useful in practice as it provides a conservative (upper bound) estimate on expected performance. The achieved accuracy should be enough for capacity planning decisions. As our model provides promising results, it may also be seen as a building block for further research on capacity planning models for search engines that may be extended to take into account other factors as discussed in the following.

A direction for future work is to extend our capacity planning model to support multiple processing threads at index servers. Another direction for research is to improve our model to estimate the distribution function of the query system response time of a cluster of index servers. This solution would be useful if the manager of the search engine requires the q-percentile of the expected query system response time to be less or equal than a given threshold.

Another direction for further research is to explicitly model the caching of query results—which allows the search engine to answer recently repeated queries at a very low cost since it is not necessary to process those queries—and caching of the inverted lists of query terms—which improves the query processing time for the new queries that include at least one term whose list is cached [35]. Identifying and analyzing the reasons for the use of disk cache, and eventually modeling the probability of disk cache hit, is also an interesting point to be addressed in future work.

Furthermore, future research may include a simulation-based analysis to verify the accuracy of our model predictions for larger clusters with thousands of index servers for supporting a collection with billions of documents, as illustrated in Section 6. Finally, another direction for further research is to develop an approach for finding the cost-optimal architecture for a vertical search engine, combining the strategies of collection partitioning and collection replication to satisfy operational requirements for query response time, query throughput, and server utilization.

8 Acknowledgments

References

[1] F. Baccelli, W. A. Massey, and D. Towsley. Acyclic fork-join queuing networks. *Journal of the ACM (JACM)*, 36(3):615–642, 1989.

[2] C. Badue, R. Baeza-Yates, W. Meira Jr., B. Ribeiro-Neto, and N. Ziviani. Distributed architecture for information retrieval. In *Proceedings of the 1st International Seminar on Advanced REsearch in E-Business (EBR´02)*, pages 114–122, Rio de Janeiro, RJ, Brazil, November 2002.

[3] C. Badue, R. Baeza-Yates, B. Ribeiro-Neto, A. Ziviani, and N. Ziviani. Analyzing imbalance among homogeneous index servers in a web search system. *Information Processing & Management (IP&M)*, 43(3):592–608, 2007.

[4] C. Badue, R. Baeza-Yates, B. Ribeiro-Neto, and N. Ziviani. Distributed query processing using partitioned inverted files. In *Proceedings of the 8th International Symposium on String Processing and Information Retrieval (SPIRE'01)*, pages 10–20, Laguna de San Rafael, Chile, 2001. IEEE Computer Society.

[5] C. Badue, R. Barbosa, P. Golgher, B. Ribeiro-Neto, and N. Ziviani. Basic issues on the processing of web queries. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*, pages 577–578, Salvador, BA, Brazil, 2005.

[6] R. Baeza-Yates. Query usage mining in search engines. In A. Scime, editor, *Web Mining: Applications and Techniques*, pages 307–321. Idea Group, 2004.

[7] R. Baeza-Yates. Applications of web query mining. In D. Losada and J.Fernández-Luna, editors, *Proceedings of the European Conference on Information Retrieval (ECIR'05)*, pages 7–22, Santiado de Compostela, Spain, 2005.

[8] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, Amsterdam, Netherlands, 2007.

[9] R. Baeza-Yates, C. Hurtado, M. Mendoza, and G. Dupret. Modeling user search behavior. In *Proceedings of the 3rd Latin American Web Congress (LA-WEB'05)*, pages 242–251, Buenos Aires, Argentina, October 2005. IEEE CS Press.

[10] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press New York, Addison Wesley, 1999.

[11] L. A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[12] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'04)*, pages 321–328, Sheffield, UK, 2004. ACM Press New York, NY, USA.

[13] E. A. Brewer. Lessons from giant-scale services. *IEEE Internet Computing*, 5(4):46–55, 2001.

[14] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[15] F. Cacheda, V. Plachouras, and I. Ounis. A case study of distributed information retrieval architectures to index one terabyte of text. *Information Processing & Management (IP&M)*, 41(5):1141–1161, 2004.

[16] I. M. Chakravarti, R. G. Laha, and J. Roy. *Handbook of Methods of Applied Statistics*, volume I, pages 392–394. John Wiley and Sons, 1967.

[17] M. Chau, X. Fang, and O. R. L. Sheng. Analysis of the query logs of a web site search engine. *Journal of the American Society for Information Science and Technology (JASIST)*, 56(13):1363–1376, 2005.

[18] R. J. Chen. A hybrid solution of fork/join synchronization in parallel queues. *IEEE Transactions on Parallel and Distributed Systems*, 12(8):829–845, 2001.

[19] A. Chowdhury and G. Pass. Operational requirements for scalable search systems. In *Proceedings of the ACM 12th Conference on Information and Knowledge Management (CIKM'03)*, pages 435–442, New Orleans, LA, USA, 2003.

[20] L. Flatto and S. Hahn. Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics*, 44(5):1042–1053, 1984.

[21] B. S. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):142–153, 1995.

[22] N. Kammenhuber, J. Luxenburger, A. Feldmann, and G. Weikum. Web search clickstreams. In *Proceedings of the 6th ACM SIGCOMM on Internet Measurement*, pages 245–250, Rio de Janeiro, RJ, Brazil, 2006. ACM Press New York, NY, USA.

[23] J. C. S. Lui, R. R. Muntz, and D. Towsley. Computing performance bounds of fork-join parallel programs under a multiprocessing environment. *IEEE Transactions on Parallel and Distributed Systems*, 9(3):295–311, 1998.

[24] A. MacFarlane, J. A. McCann, and S. E. Robertson. Parallel search using partitioned inverted files. In *Proceedings of the 7th International Symposium on String Processing and Information Retrieval (SPIRE'00)*, pages 209–220, La Coruña, Spain, 2000. IEEE Computer Society.

[25] D. A. Menascé, V. A. F. Almeida, and L. W. Dowdy. *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall, 2004.

[26] A. Moffat, W. Webber, and J. Zobel. Load balancing for term-distributed parallel retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'06)*, pages 348–355, Seattle, WA, USA, 2006.

[27] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10(3):205–231, 2007.

[28] R. Nelson and A. N. Tantawi. Approximate analysis of fork/join synchronization in parallel queues. *IEEE Transactions on Computers*, 37(6):739–743, 1988.

[29] A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*, pages 191–198, Amsterdam, The Netherlands, 2007.

[30] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queuing networks. *Journal of the ACM (JACM)*, 27(2):313–322, 1980.

[31] B. A. Ribeiro-Neto and R. A. Barbosa. Query performance for tightly coupled distributed digital libraries. In *Proceedings of the 3rd ACM Conference on Digital Libraries*, pages 182–190, Pittsburgh, PA, USA, 1998.

[32] K. M. Risvik, Y. Aasheim, and M. Lidal. Multi-tier architecture for web search engines. In *Proceedings of the 1st Latin American Web Congress (LA-WEB'03)*, pages 132–143, Santiago, Chile, November 2003.

[33] D. Rose and D. Levinson. Understanding user goals in web search. In *Proceedings of the 13th International World Wide Web Conference (WWW'04)*, pages 13–19, New York, NY, USA, 2004. ACM Press.

[34] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[35] P. C. Saraiva, E. S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Ribeiro-Neto. Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ´01)*, pages 51–58, New Orleans, Louisiana, USA, 2001.

[36] C. Silverstein, M. Henzinger, H. Marais, and M. Moriez. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[37] O. Sornil and E. A. Fox. Hybrid partitioned inverted indices for large-scale digital libraries. In *Proceedings of the 4th International Conference of Asian Digital Libraries*, Bangalore, India, 2001.

[38] A. Spink, B. J. Jansen, and H. C. Ozmultu. Use of query reformulation and relevance feedback by excite users. *Internet Research: Electronic Networking Applications and Policy*, 10(4):317–328, 2000.

[39] A. Thomasian and A. N. Tantawi. Approximate solutions for M/G/1 fork/join synchronization. In *Proceedings of the 26th Conference on Winter Simulation (WSC '94)*, pages 361–368, San Diego, CA, USA, 1994. Society for Computer Simulation International.

[40] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems (PDIS'93)*, pages 8–17, San Diego, CA, USA, 1993.

[41] E. Varki. Mean value technique for closed fork-join networks. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '99)*, pages 103–112, New York, NY, USA, 1999. ACM Press.

[42] E. Varki. Response time analysis of parallel computer and storage systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(11):1146–1161, 2001.

[43] E. Varki and L. W. Dowdy. Analysis of balanced fork-join queueing networks. In *Proceedings of the 1996 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 232–241, Philadelphia, PA, USA, 1996.

[44] S. Varma and A. M. Makowski. Interpolation approximations for symmetric fork-join queues. In *Performance Evaluation*, pages 245–265, 1994.

[45] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, Inc., 2nd edition, 1999.

[46] K. H. Yeung and T. S. Yum. State reduction in the exact analysis of fork/join queueing systems with homogeneous exponential servers. In *Proceedings of the 1996 International Conference on Parallel and Distributed Systems (ICPADS'96)*, pages 57–, 1996.

[47] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys (CSUR)*, 38(2):6–es, 2006.